

Theme I

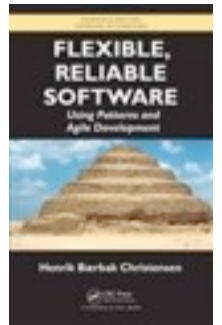
Software Processes

Focus on Extreme Programming

Table of content

- Definition eXtreme Programming
- Basic values
- Principles
- Practices
 - pair programming
 - stories
 - test-first programming
 - continuous integration

Extreme Programming (XP)



- Point of XP: coping with change and uncertainty
- **software product = $f(c,t,s,q)$**
- parameters:
 - cost \approx number of staff on the project
 - time \approx time to delivery deadline
 - scope \approx amount of functionality
 - quality \approx reliability

Extreme Programming



- Software Product = $f(c, t, s, q)$

- **Question:**

Which parameters are fixed by the management?



- **Question:**

Which parameter is left for developers to tweak in order to adjust the workload?

Extreme Programming



- Software Product = $f(c, t, s, q)$
- **Question:**
Which parameters are fixed by the management?
Cost, Time, Scope
- **Question:**
Which parameter is left for developers to tweak in order to adjust the workload?
Quality

Extreme Programming

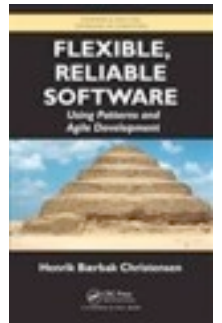
- The Agile/XP answer:

Projects fix [price, time, quality]

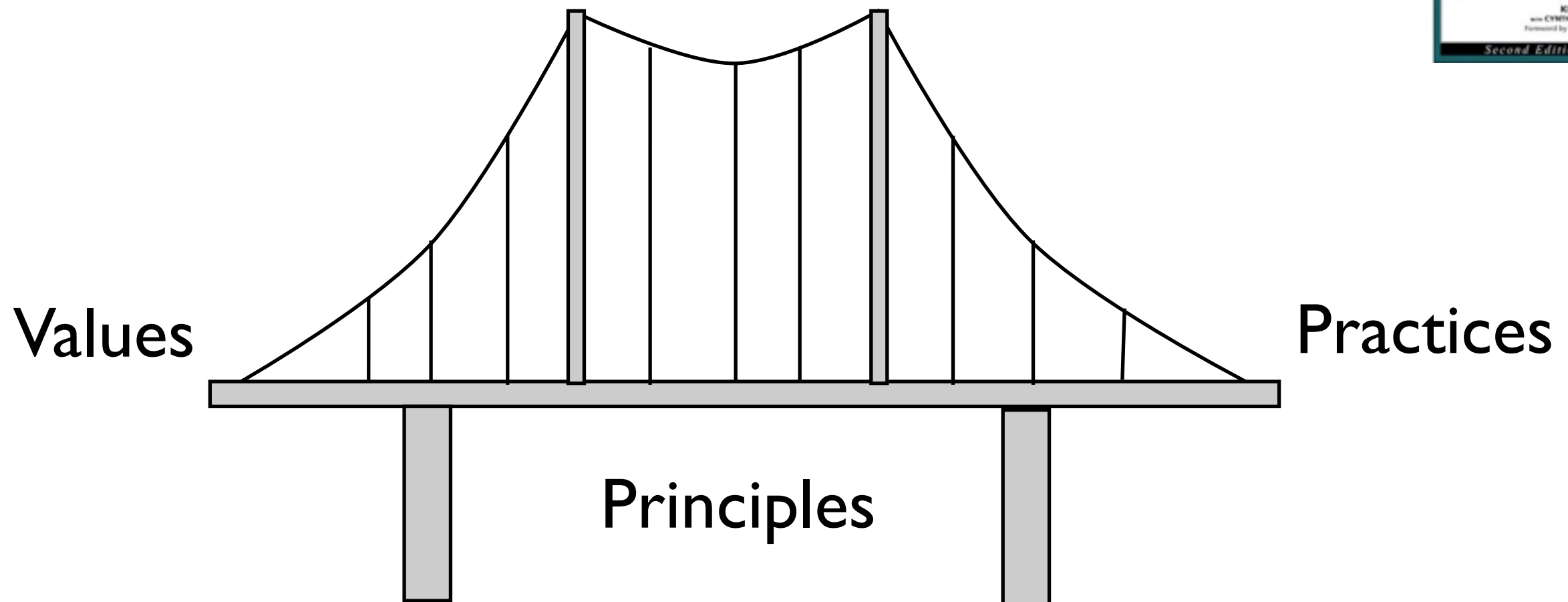
Open parameter [scope]

Meaning:

- Scope control important
 - On site customer / user feedback
 - Rapid release cycles
- Quality control important
 - Testing & user feedback
 - Test-driven development



Describing XP



Basic XP Values



- Communication
 - communicate problems&solutions, teamwork
- Simplicity
 - eliminate wasted complexity
- Feedback
 - change creates the need for feedback
- Courage
 - effective action in the face of fear
- Respect
 - care about you, the team, and the project

Principles



Humanity, Economics, Mutual Benefit, Self-Similarity, Improvement, Diversity, Reflection, Flow, Opportunity, Redundancy, Failure, Quality, Baby Steps, Accepted Responsibility

Will not detail them -- they govern what the practices tend to accomplish

So, on to the practices!

Primary Practices

Sit Together

Whole Team

Energized Work

Pair Programming

Stories

Automated Testing

Small Releases

Weekly Cycle

Quarterly Cycle

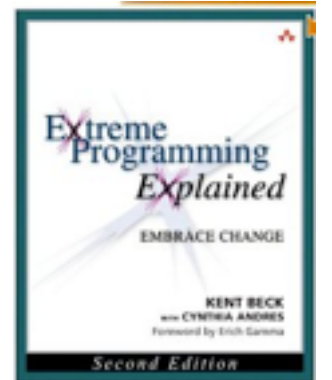
Ten Minute Build

**Continuous
Integration**

Incremental Design

**Collective Code
Ownership**

Coding Standards



Pair Programming

- Write all production programs with two people sitting at one machine
 - make enough room, move keyboard and mouse
- Pair programmers:
 - keep each other on task
 - brainstorm refinements to the system
 - clarify ideas
 - take initiative when partner is stuck (less frustration)
 - hold each other accountable to practices

Pair programming and privacy

- Sometimes you might need some privacy
 - then go work alone
 - come back with the idea (NOT the code)
 - quickly reimplemented with two
 - benefits the whole team, not you alone
- Rotate pairs frequently
 - every couple of hours, at natural breaks in development
 - with a timer, every 60 minutes (or 30 minutes for difficult problems)

Stories

plan using units of customer-visible functionality

name

estimate

Save with compression

8 hrs

Currently the compression options are in a dialog subsequent to the save dialog. Make them part of the save dialog itself

short description

index card

7 more User Stories

- Students can purchase monthly parking passes online.
- Parking passes can be paid via credit cards.
- Parking passes can be paid via PayPal TM.
- Professors can input student marks.
- Students can obtain their current seminar schedule.
- Students can only enroll in seminars for which they have prerequisites.
- Transcripts will be available online via a standard browser.

Another example

173. Students can purchase parking passes.

Priority: 8

Estimate: 4

User Stories vs. Use Case

- Not the same artifact!

Continuous Integration

- Team Programming = Divide, Conquer, *Integrate*
- Integrate and test changes after no more than a couple of hours
 - integration typically takes long
 - when done at the end, risks the whole project when integration problems are discovered
 - the longer you wait, the more it costs and the more unpredictable it becomes

Using Continuous Integration

- Synchronous
 - After a task is finished, you integrate and run the tests
 - Immediate feedback for you and your partner
- Asynchronous
 - After submitting changes, the build system notices something new, builds and tests the system, and gives feedback by mail, notification, etc.
 - Feedback typically comes when a new task is started
 - Pair programmers might have been switched already

Test-first Programming

- Write a failing automated test before changing code
- Addresses many problems:
 - **Scope creep**: focus coding by what the code should do, not on the “just in case” code
 - **Coupling and cohesion**: If it’s hard to write a test, there is a design problem (not a testing problem)
 - **Trust**: clean working code + automated tests
 - **Rhythm**: gives focus on what to do next
 - efficient rhythm: test, code, refactor, test, ...
- See next theme!!!

Conclusion XP

- It is a **process** that is
 - **incremental**: growing software instead of designing
 - **iterative**: learning while doing.